Devoir surveillé n° 03 — corrigé

EXERCICE 1 *Trace de l'algorithme de Kosaraju-Sharir*

On obtient les intervalles pour le premier parcours dans le graphe miroir

$$A : [0,3]; B : [4,15]; C : [9,10]; D : [1,2]; E : [5,14]; F : [8,11]; G : [6,13]; H : [7,12]; I : [16,17]$$

Les sommets triés dans l'ordre inverse de leur temps de fin de parcours est alors

$$I, b, E, G, H, F, C, A, D$$

Un deuxième parcours du graphe initial en suivant cet ordre donne les intervalles suivants

$$A : [14,15]; B : [2,13]; C : [3,12]; D : [16,17]; E : [7,8]; F : [4,12]; G : [6,9]; H : [5,10]; I : [0,1]$$

Il y a 5 arcs de parcours, 3 arcs arrières, aucun arc avant et 4 arcs transverses.

EXERCICE 2

On utilise trois status possibles pour le tableau de marques : `V` pour *vierge*, `A` pour *actif* et `F` pour *fini*. On explore récursivement à partir de chaque sommet, en explorant récursivement tous ses voisins vierges et en comptant le nombre de voisins actifs, qui correspondent aux arcs arrières.

OCAML

```
type status = V | A | F

let dfs g =
  let n = size g in
  let s = Array.make n V in
  let p = Array.make n (-1) in
  let r = ref 0 in
  let rec explore px x =
    match s.(x) with
    | F -> ()
    | A -> incr r
    | V ->
       s.(x) <- A;
       p.(x) <- px;
       List.iter (explore x) (neighbors g x);
       s.(x) <- F
  in
  for x = 0 to n - 1 do
    explore x x
  done;
  !r, p
```

EXERCICE 3 *Composantes connexes par unir et trouver*

Les deux premières questions sont vraiment des questions de cours.

1. Chaque élément est racine d'un arbre de rang 0 ne contenant que lui-même.

OCaml
```
let creer_singletons n =
  {pere = Array.init n (fun i -> i); rang = Array.make n 0}
```

2. On cherche récursivement le représentant en mettant à jour au passage.

OCaml
```
let rec trouver p x =
  if p.pere.(x) <> x then
    p.pere.(x) <- trouver p p.pere.(x);
  p.pere.(x)
```

La fonction `unir` n'était pas demandé mais nous la donnons ici pour mémoire :

OCaml
```
let unir p x y =
  let rx = trouver p x in
  let ry = trouver p y in
  if rx <> ry then begin
    if p.rang.(rx) < p.rang.(ry) then
      p.pere.(rx) <- ry
    else begin
      p.pere.(ry) <- rx;
      if p.rang.(rx) = p.rang.(ry) then
        p.rang.(rx) <- p.rang.(rx) + 1
    end
  end
```

3. Il suffit simplement d'unir tous les sommets reliés par une arête. D'après l'énoncé, comme il y a $|S| + |A|$ opérations sur la structure, la complexité est en $O((|S| + |A|)\alpha(|S| + |A|))$.

OCaml
```
let composantes_connexes g =
  let p = creer_singletons g.nb_sommets in
  List.iter (fun (x, y) -> unir p x y) g.aretes;
  p
```

4. On pourrait transformer la représentation du graphe sous forme de liste d'arêtes en représentation par tableau de listes d'adjacence en $O(|S| + |A|)$, puis identifier les composantes connexes par l'algorithme classique qui effectue un parcours en $O(|S| + |A|)$.
5. Dans la première approche, il suffit de faire un seul appel à `unir`. La complexité peut-être en $O(\log(|S| + |A|)$ mais on a la garantie de l'énoncé sur la complexité amortie, donc ceci ne peut pas se produire trop souvent. L'utilisation de la structure de données *unir et trouver* est donc avantageuse pour un graphe dont les arêtes ne sont connues qu'au fur et à mesure. Avec la deuxième approche il faut parcourir entièrement une des deux composantes pour remettre à jour l'identifiant, ce qui peut être en $O(|S| + |A|)$.